

EXHIBIT 5

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC.,

Plaintiff,

v.

GOOGLE INC.,

Defendant.

Case No. CV 10-03561 WHA

**EXPERT REPORT OF ROBERT
ZEIDMAN**

Table of Contents

I.	Experience and Background	1
II.	Reliance Materials	2
A.	Java Source Code	2
B.	Android Source Code	3
C.	Software Tools	7
1.	Software Tools.....	7
III.	Background	8
A.	Software Source Code.....	8
B.	Java APIs.....	9
IV.	Assignment	9
V.	Summary of Opinions.....	10
VI.	Analysis.....	12
A.	Google Copied Declaring Code for the 37 Java API Packages from Java SE 5 in Android Gingerbread (API Level 9) and Android Lollipop (API Level 22).....	12
1.	Declaring Code	12
2.	Script Used to Identify Java Declaring Code Copied in Android	16
3.	Process	18
4.	Presence of Private Java Declarations.....	21
B.	Google Copied Declaring Code for the 37 Java API Packages from Java SE 5 in Other Versions of Android.....	22
1.	Source Code Analysis of Other Versions of Android	23
2.	Google’s Public Documents Confirm Copying Exists in Other Versions of Android.....	25
3.	Google Copied Declaring Code from Java SE 5 in Android Honeycomb.	30
C.	Google Copied the SSO of the 37 Java API Packages From Java SE 5 in Android.....	31
D.	Google Has Continued to Copy New Declaring Code and SSO of the 37 Java API Packages from Java SE 6 and Java SE 7.....	32
E.	Google Copied the Declaring Code and SSO of the 37 Java API Packages in Google’s “ARC” Product	33
1.	The App Runtime for Chrome (“ARC”) Technology Allows Android Apps to Run on Chrome OS Devices.....	33
2.	The ARC Runtime Introduces Android Runtime Components into Chrome OS Devices.....	34
3.	The ARC Runtime Build Process Contains the 37 Java API Packages.	36
4.	Android Apps Made Available for Use on Chrome through ARC Rely on and Use the 37 Java API Packages	37

VII. Conclusion	38
-----------------------	----

Exhibit	Description
A	Resume
B	Materials Relied Upon
C	List of 37 Java API Packages
D	DECLARATION_PROCESSOR.py
E	AndroidDeclarationDiff.py
F	Android Gingerbread (API Level 9) Analysis
G	Android Gingerbread (API Level 10) Analysis
H	Android Honeycomb (API Level 11) Analysis
I	Android Honeycomb (API Level 12) Analysis
J	Android Honeycomb (API Level 13) Analysis
K	Android Ice Cream Sandwich (API Level 14) Analysis
L	Android Ice Cream Sandwich (API Level 15) Analysis
M	Android Jelly Bean (API Level 16) Analysis
N	Android Jelly Bean (API Level 17) Analysis
O	Android Jelly Bean (API Level 18) Analysis
P	Android Kit Kat (API Level 19) Analysis
Q	Android Kit Kat (API Level 20) Analysis
R	Android Lollipop (API Level 21) Analysis
S	Android Lollipop (API Level 22) Analysis
T	Android Marshmallow (API Level 23) Analysis
U	Side-by-Side Comparison of API Documentation
V	Analysis of Java SE 6
W	Analysis of Java SE 7
X	readelf Output
Y	Diff Report for Android Gingerbread (API Level 9)
Z	Diff Report for Android Honeycomb (API Level 11)
AA	Diff Report for Android Ice Cream Sandwich (API Level 14)
BB	Diff Report for Android Kit Kat (API Level 19)
CC	Diff Report for Android Lollipop (API Level 21)

I. EXPERIENCE AND BACKGROUND

1. I am an engineer and the founder and president of Zeidman Consulting, which provides engineering consulting to high-tech companies. Among the types of services I provide are hardware and software design. My clients have included Fortune 500 computer and technology companies as well as smaller companies and startups. A copy of my resume is attached hereto as Exhibit A.

2. I hold a Master's degree from Stanford University in Electrical Engineering and two Bachelor's degrees from Cornell University, one in Electrical Engineering and one in Physics.

3. I have been a computer software and hardware designer for over 30 years, having designed and developed a variety of computer hardware and software. The software products include Internet-based training courses and web-based course administration software, an operating system synthesis tool, a source code comparison tool, a network emulation software bridge, and a remote backup system whereby user data is automatically transmitted and stored at a remote location. I have founded several companies including Zeidman Consulting, a hardware and software development firm, eVault, a remote backup company, the Chalkboard Network, an e-learning company, Zeidman Technologies, that develops software tools for enabling and improving hardware and software development, and Software Analysis and Forensic Engineering Corporation that develops software analysis tools.

4. I have written a variety of papers, books, and presentations on computer hardware and software and other engineering subjects. I am the developer of the Universal Design Methodology, a process for efficiently developing reliable systems, about which I have written extensively. A list of my publications is included in my resume attached as Exhibit A.

5. I hold 22 patents in the areas of software analysis, software comparison, software synthesis, hardware emulation, hardware synthesis, hardware simulation, and media broadcast and advertising. I have created a tool called CodeSuite[®] for assisting in the determination of whether one computer program has been copied from another computer program.

6. I have consulted on matters involving intellectual property disputes, including instances of alleged software copying, trade secret misappropriation, and patent infringement. My work in this capacity has included, among other things, reviewing and analyzing software source code, reviewing and analyzing patents, reverse engineering hardware and software, writing expert reports, and testifying in court.

7. I have extensive experience writing and analyzing Java code. For my e-learning company The Chalkboard Network, I took over full responsibility for the backend course management software that was written in Java, making all bug fixes, modifications, and functionality upgrades. I created the prototype of the SynthOS software development tool that is developed and sold by my company Zeidman Technologies. I wrote that prototype from scratch in Java. Also, I have worked as a forensic engineer on many litigations where I was required to analyze Java code.

8. I have testified at deposition and at trial in a number of cases including software copyright infringement, trade secret theft, and patent infringement. I have testified in federal court on eight occasions. The specific cases can be found in my resume, attached as Exhibit A.

9. For my work on this matter Zeidman Consulting is being compensated at a rate of \$800 per hour. My compensation is not contingent on the nature of my findings, on my testimony if I am called to testify, or on the outcome of this litigation.

10. I am being supported in this matter by consultants employed by Zeidman Consulting. Zeidman Consulting is being compensated for these consultants at rates less than \$800 per hour.

II. RELIANCE MATERIALS

11. A full list of materials I relied upon in reaching the opinions expressed in this reports is attached as Exhibit B. These materials include:

A. Java Source Code

12. For the Java platform, I analyzed source code from the Java Development Kit

(“JDK”) and Java Runtime Environment (“JRE”) for Java SE 5.¹

13. I understand that the source code for Java SE 5 was Trial Exhibit 623 during the first trial. I received this exhibit as a .zip file from counsel (0623.zip).

14. I focused my analysis of Java source code to 37 Java API Packages owned by Oracle that were at issue in the first trial. The source code for the 37 Java API Packages within Java SE 5 was located in the following directories:

1. \0623.zip\0623\OAGOOGL0100209734_HIGHLY CONFIDENTIAL - SOURCE CODE\licensebundles\jdk1.5.0\jdk-1_5_0-fcs-bin-b64-windows-i586-15_sep_2004\jdk1.5.0\src.zip\java\
2. \0623.zip\0623\OAGOOGL0100209734_HIGHLY CONFIDENTIAL - SOURCE CODE\licensebundles\jdk1.5.0\jdk-1_5_0-fcs-bin-b64-windows-i586-15_sep_2004\jdk1.5.0\src.zip\javax\
3. \0623.zip\0623\OAGOOGL0100209734_HIGHLY CONFIDENTIAL - SOURCE CODE\licensebundles\source-bundles\jdk-1_5_0-src-windows\j2se\src\share\doc\stub\javax\

B. Android Source Code

15. Google gives each major release of Android a dessert-themed code name. For each of these codenames there is one or more numbered Android versions. Each Android version has an associated API Level. The API Level is a revision or version of the Android APIs.² Table 1 summarizes the different versions of Android and the different API Levels.

Codename	Android Version	API Version (API Level)	Release Date
	Android 1.0	API Level 1	Sept. 2008
	Android 1.1	API Level 2	Feb. 2009
Cupcake	Android 1.5	API Level 3	Apr. 2009

¹ Versions 1.5.0 and 5.0 of the Java 2 Platform Standard Edition are the same. *See* <http://docs.oracle.com/javase/1.5.0/docs/relnotes/version-5.0.html>.

² <http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels>

Codename	Android Version	API Version (API Level)	Release Date
Donut	Android 1.6	API Level 4	Sept. 2009
Eclair	Android 2.0	API Level 5	Oct. 2009
	Android 2.0.1	API Level 6	Dec. 2009
	Android 2.1	API Level 7	Jan. 2010
Froyo	Android 2.2.x	API Level 8	May 2010
Gingerbread	Android 2.3–2.3.2	API Level 9	Dec. 2010
	Android 2.3.3–2.3.7	API Level 10	Feb. 2011
Honeycomb	Android 3.0	API Level 11	Feb. 2011
	Android 3.1	API Level 12	May 2011
	Android 3.2.x	API Level 13	Jul. 2011
Ice Cream Sandwich	Android 4.0–4.0.2	API Level 14	Oct. 2011
	Android 4.0.3–4.0.4	API Level 15	Dec. 2011
Jelly Bean	Android 4.1.x	API Level 16	Jul. 2012
	Android 4.2.x	API Level 17	Nov. 2012
	Android 4.3.x	API Level 18	Jul. 2013
Kit Kat	Android 4.4–4.4.4	API Level 19	Oct. 2013
	Android 4.4W–4.4W.2	API Level 20	Jul. 2014
Lollipop	Android 5.0	API Level 21	Nov. 2014
	Android 5.1	API Level 22	Mar. 2015
Marshmallow	Android 6.0	API Level 23	Oct. 2015

Table 1. List of Android Codenames, versions, and API Levels. See <http://source.android.com/source/build-numbers.html>.

16. My understanding is that the first trial in this case addressed Google’s copying of the 37 Java API Packages through Android Froyo (API Level 8), therefore my analysis of Android source code in this report focuses on Android Gingerbread (API Level 9) and later. Specifically, I relied on:

- a. The Android source code for Android Gingerbread (API Levels 9, 10), Android Ice Cream Sandwich (API Levels 14, 15), Android Jelly Bean (API Levels 16, 17, 18) Android Kit Kat (API Level 19), Android Lollipop (API Levels 21, 22), and Android Marshmallow (API Level 23). This source code was downloaded by following instructions found at <https://source.android.com/source/downloading.html>.
- b. The Android Source code for Android Kit Kat (API Level 20). This source code was downloaded via the Android SDK Manager, a development tool Google makes available on its Android Developer website. *See*

<http://developer.android.com/sdk/index.html>.

17. Within Android Gingerbread (API Level 9), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

`\API9_Gingerbread\libcore\luni\src\main\java\java\
\API9_Gingerbread\libcore\luni\src\main\java\javax\
\API9_Gingerbread\libcore\luni\src\main\java\java\
\API9_Gingerbread\libcore\luni\src\main\java\javax\
\API9_Gingerbread\libcore\luni\src\main\java\java\
\API9_Gingerbread\libcore\luni\src\main\java\javax`

18. Within Android Gingerbread (API Level 10), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

`\API10_GingerbreadMR1\libcore\luni\src\main\java\java\
\API10_GingerbreadMR1\libcore\luni\src\main\java\javax\
\API10_GingerbreadMR1\libcore\luni\src\main\java\java\
\API10_GingerbreadMR1\libcore\luni\src\main\java\javax`

19. Within Android Ice Cream Sandwich (API Level 14), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

`\API14_IceCreamSandwich\libcore\luni\src\main\java\java\
\API14_IceCreamSandwich\libcore\luni\src\main\java\javax\
\API14_IceCreamSandwich\libcore\luni\src\main\java\java\
\API14_IceCreamSandwich\libcore\luni\src\main\java\javax`

20. Within Android Jellybean (API Level 15), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

`\API15_IceCreamSandwichMR1\libcore\luni\src\main\java\java\
\API15_IceCreamSandwichMR1\libcore\luni\src\main\java\javax\
\API15_IceCreamSandwichMR1\libcore\luni\src\main\java\java\
\API15_IceCreamSandwichMR1\libcore\luni\src\main\java\javax`

21. Within Android Jellybean (API Level 16), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

`\API16_JellyBean\libcore\luni\src\main\java\java\
\API16_JellyBean\libcore\luni\src\main\java\javax\
\API16_JellyBean\libcore\luni\src\main\java\java\
\API16_JellyBean\libcore\luni\src\main\java\javax`

22. Within Android Jelly Bean (API Level 17), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

`\API17_JellyBeanMR1\libcore\luni\src\main\java\java\
\API17_JellyBeanMR1\libcore\luni\src\main\java\javax\
\API17_JellyBeanMR1\libcore\luni\src\main\java\java\
\API17_JellyBeanMR1\libcore\luni\src\main\java\javax`

23. Within Android Jelly Bean (API Level 18), I specifically analyzed and relied upon

source code in the following directories where I found source code for the 37 Java API Packages:

```
\API18_JellyBeanMR2\libcore\luni\src\main\java\java\  
\API18_JellyBeanMR2\libcore\luni\src\main\java\javax\
```

24. Within Android Kit Kat (API Level 19), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

```
\API19_KitKat\libcore\luni\src\main\java\java\  
\API19_KitKat\libcore\luni\src\main\java\javax\  
\API19_KitKat\libcore\libart\src\main\java\java\
```

25. Within Android Kit Kat (API Level 20), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

```
\android-sdk\sources\android-20\java\  
\android-sdk\sources\android-20\javax\
```

26. Within Android Lollipop (API Level 21), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

```
\API21_Lollipop\libcore\luni\src\main\java\java\  
\API21_Lollipop\libcore\luni\src\main\java\javax\  
\API21_Lollipop\libcore\libart\src\main\java\java\
```

27. Within Android Lollipop (API Level 22), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

```
\API22_LollipopMR1\libcore\luni\src\main\java\java\  
\API22_LollipopMR1\libcore\luni\src\main\java\javax\  
\API22_LollipopMR1\libcore\libart\src\main\java\java\
```

28. Within Android Marshmallow (API Level 23), I specifically analyzed and relied upon source code in the following directories where I found source code for the 37 Java API Packages:

```
\API23_Marshmallow\libcore\luni\src\main\java\java\
```

```
\API23_Marshmallow\libcore\luni\src\main\java\javax\  
\API23_Marshmallow\libcore\libart\src\main\java\java\
```

29. These versions of Android analyzed and relied upon include those related to Google's Android Auto, Android TV, and Android Wear initiatives. Android apps designed for either Android Auto or Android TV must be compatible with, at a minimum, Android Lollipop (API Level 21). <http://developer.android.com/training/auto/start/index.html>; <http://developer.android.com/training/tv/start/start.html>. Android apps designed for Android Wear must be compatible with, at a minimum, Android Kit Kat (API Level 20). <http://developer.android.com/training/wearables/apps/creating.html>.

30. Absent from this list of Android source code I relied upon is the source code for Android Honeycomb (API Level 11 through 13). I was unable to download the source code from Google's online source code repository and I understand it is not available for download. <https://source.android.com/source/build-numbers.html#honeycomb-gpl-modules>. I was also unable to retrieve the source code from a hard drive Google produced in this case that purports to be a mirror of Google's Android Git repository because it appears files were missing that were required to use the Git repository. Instead of identifying copying of Java SE 5 declaring code in Android Honeycomb by analyzing source code, I achieved the same result using Google's own public statements. *See* Paragraph 108 through 111, below.

C. Software Tools

1. *Software Tools*

31. I relied upon numerous software tools as part of my analysis, including:

a. Understand

32. Understand is a commercial tool from Scientific Toolworks that analyzes and provides pertinent information regarding a collection of code. Specifically, Understand identifies all entities, which include fields, methods, constructors, parameters, classes, interfaces,

and annotations, and creates a Data Dictionary Report that can be used for further analysis. I used Understand to identify and output potential declaring code within both the Java and Android code bases that I reviewed. I also manually searched for package and import statements.

b. Cygwin

33. Cygwin is open-source software that creates a Linux command line environment in Windows. I used Cygwin to run Python scripts.

c. Python

34. Python is a widely used open source scripting language that has many built-in functions for parsing, searching, and managing strings. I used Python to compare source code from Java and Android.

III. BACKGROUND

35. This section describes the technological issues relevant as background to understand my analysis and opinion.

A. Software Source Code

36. Computer programs can be written using complex instructions that look like English. For example, the instruction `a = b*c+2` tells the computer to take the number stored in memory and represented by variable `b`, multiply that by the number stored in memory and represented by the variable `c`, add 2 and store the result in memory represented by the variable `a`. Similarly, the statement `printf("Hello world!")` tells the computer to print the words "Hello world!" to the computer screen. These high-level, English-like instructions are called "source code." Computer programs are made up of many lines of source code and the process of writing these lines of code is called programming. Eventually these lines of source code are turned into instructions that a computer understands, consisting of sequences of electronic ones and zeroes. The process of turning human-readable source code into a file containing computer instructions

is called “compiling” and is performed by a special computer program called a “compiler.” In some cases, source code is run directly by a computer, without creating any file of computer instructions, in which case the program is “interpreted” by a special computer program called an “interpreter” that converts each line of source code to computer instructions one at a time to be executed by the computer. Java source code is compiled into byte code, which is interpreted by the Java Virtual Machine.

B. Java APIs

37. The Java Platform includes the Java API Specification, *see, e.g.*, <http://docs.oracle.com/javase/1.5.0/docs/api/index.html>, a set of prewritten classes and interfaces with their respective methods, fields, and constructors. I understand these prewritten programs, generally referred to as Java APIs, assist programmers to write their own programs and are organized into “packages.”

IV. ASSIGNMENT

38. On or about July 27, 2015, Oracle America, Inc. (“Oracle”) engaged me to analyze the source code that comprises the Java API Specification and Android API Specification to identify instances where Google copied declaring code from the Java Platform into various versions of the Android Platform.

39. I understand that there was a trial in this matter in April and May 2012 that focused in part on 37 Java API Packages that Google copied in various versions of Android. I also understand that after a jury verdict in Oracle’s favor on the issue of infringement, the parties appealed to the United States Court of Appeals for the Federal Circuit (“Federal Circuit”), wherein the Federal Circuit acknowledged that “Google conceded that it copied the declaring code used in the 37 packages verbatim.” *Oracle America Inc. v. Google Inc.*, 750 F.3d 1339, 1351 (Fed Cir. 2014).

40. My assignment included analyzing Android source code to identify instances where

Google copied declaring code and sequence, structure, and organization (“SSO”) of the 37 Java API Packages from the Java Platform to Android. To this end, I analyzed Android Gingerbread (API Level 9), the first version of Android not analyzed at the previous trial, through Android Lollipop (API Level 22), the most recent version of Android released at the time I started my analysis. Since I started my analysis, Google released Android Marshmallow (API Level 23) and I include this version in my analysis.

41. I also looked for evidence of copying of Oracle’s copyrighted code in other products, including a Chromebook running Chrome OS.

42. I also plan to prepare demonstratives and summaries relevant to my opinions if called to testify in this case.

V. SUMMARY OF OPINIONS

43. I have analyzed the source code for Java SE 5 and Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23), looking for instances where declaring code for the 37 Java API Packages from Java SE 5 appears in Android. Based on this analysis, I conclude the following:

- a. Android Gingerbread (API Level 9) contains 11,723 declaring code statements from Java SE 5;
- b. Android Gingerbread (API Level 10) contains 11,723 declaring code statements from Java SE 5;
- c. Android Honeycomb (API Level 11) contains 11,723 declaring code statements from Java SE 5;
- d. Android Honeycomb (API Level 12) contains 11,723 declaring code statements from Java SE 5;
- e. Android Honeycomb (API Level 13) contains 11,723 declaring code statements from Java SE 5;

- f. Android Ice Cream Sandwich (API Level 14) contains 11,692 declaring code statements from Java SE 5;
 - g. Android Jelly Bean (API Level 15) contains 11,692 declaring code statements from Java SE 5;
 - h. Android Jelly Bean (API Level 16) contains 11,693 declaring code statements from Java SE 5;
 - i. Android Jelly Bean (API Level 17) contains 11,694 declaring code statements from Java SE 5;
 - j. Android Jelly Bean (API Level 18) contains 11,720 declaring code statements from Java SE 5;
 - k. Android Kit Kat (API Level 19) contains 11,730 declaring code statements from Java SE 5;
 - l. Android Kit Kat (API Level 20) contains 11,731 declaring code statements from Java SE 5;
 - m. Android API Lollipop (Level 21) contains 11,734 declaring code statements from Java SE 5;
 - n. Android Lollipop (API Level 22) contains 11,735 declaring code statements from Java SE 5;
 - o. Android Marshmallow (API Level 23) contains 11,717 declaring code statements from Java SE 5;
44. I further conclude that Google copied the SSO for the 37 Java API Packages in Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23).
45. I also conclude that Google copied 816 lines of declaring code from Java SE 6 and 223 lines of declaring code from Java SE 7 in Android Gingerbread (API Level 9) through

Android Marshmallow (API Level 23).

46. Finally, I conclude that by enabling Android apps to be run on Chrome OS devices such as Chromebooks, Google has copied the declaring code and SSO of the 37 Java API Packages.

VI. ANALYSIS

A. Google Copied Declaring Code for the 37 Java API Packages from Java SE 5 in Android Gingerbread (API Level 9) and Android Lollipop (API Level 22)

47. The following sections describe the tools and procedures I used to perform an analysis to identify instances where Google copied the declaring code and SSO from the 37 Java API Packages in Android. I performed the analysis described below on two versions of Android: (1) Android Gingerbread (API Level 9) and Android Lollipop (API Level 22). I also performed another analysis, as described in Section VI.B, to confirm that Google also copied declaring code and SSO in other versions of Android.

48. Note that the source code files for the 37 Java API Packages at issue are organized in the same directories and files in Android as they are in Java. For example, package `java.lang.reflect` was found in the Java directory path `\jdk1.5.0\src\java\lang\reflect\` and in the Android directory path `\libcore\luni\src\main\java\java\lang\reflect` where the last part of the directory paths was identical. These last parts of the directory paths represent the package names.

1. Declaring Code

49. The source code analysis focused on “declaring code” in the Java and Android platforms. “Declaring code” refers to source code that introduces or specifies an entity in the Java Platform. This includes declarations for packages, imports, classes, fields, methods, constructors, interfaces, and annotations. A summary of the various types of declaring code and

criteria determining that Google copied declaring code is contained in Table 2. Ascertaining the criteria for identifying source code that was possibly copied is an important first step before beginning a code review and one I regularly use in my work.

Declaring Code Type	Criteria For Identifying Copied Declaring Code
Package statements	Identical package name
Import statements	Identical packages imported
Class Declarations	Identical name Extends identical classes Implements identical interfaces, regardless of order Identical explicit or implicit access modifiers
Field Declarations	Identical name and type Identical explicit or implicit access modifiers
Method Declarations	Identical name and return type Identical number, order, and type of parameters Identical exceptions thrown Identical explicit or implicit access modifiers
Constructor Declarations	Identical name Identical number, order, and type of parameters Identical exceptions thrown Identical explicit or implicit access modifiers
Interface Declarations	Identical name Extends identical interfaces Identical explicit or implicit access modifiers
Annotation	Identical name Identical explicit or implicit access modifiers

Table 2. Declaring code

a. Package Statements.

50. The criterion I used for identifying package statements in Android that Google copied from Java is that the package statements met the criteria in Table 2.

51. An example of a package statement Google copied from Java is in the

`AbstractInterruptibleChannel.java` file in the `java.nio.channels.spi` package on line 10 of the Java SE 5 and line 18 of the Android Lollipop (API Level 22) source code, respectively:

Java SE 5: `package java.nio.channels.spi;`

Android Lollipop (API Level 22): `package
java.nio.channels.spi;`

b. Import Statements.

52. The criterion I used for identifying import statements in Android that Google copied from Java is that the import statements met the criteria in Table 2.

53. An example of an import statement Google copied from Java is in the `x509Certificate.java` file in the `java.security.cert` package on line 16 of the Java SE 5 and line 26 of the Android Lollipop (API Level 22) source code, respectively:

Java SE 5: `import
javax.security.auth.x500.X500Principal;`

Android Lollipop (API Level 22): `import
javax.security.auth.x500.X500Principal;`

c. Class Declarations.

54. I used the criteria in Table 2 to identify class declarations in Android that Google copied from Java.

55. An example of a class declaration Google copied from Java is in the `CharBuffer.java` file in the `java.nio` package on lines 240-242 and lines 38-39 of the Java SE 5 and API Level 22 (Android Lollipop) source code, respectively:

Java SE 5: `public abstract class CharBuffer extends
Buffer implements Comparable<CharBuffer>,
Appendable, CharSequence, Readable`

Android Lollipop (API Level 22): `public abstract class
CharBuffer extends Buffer implements
Comparable<CharBuffer>, CharSequence, Appendable,
Readable`

56. Another example of a class declaration Google copied from Java is in the

`Collections.java` file of the `java.util` package on lines 3054-3056 and line 223 of the Java SE 5 and API Level 22 (Android Lollipop) source code, respectively:

```
Java SE 5: private static class SingletonSet<E>
    extends AbstractSet<E> implements Serializable

Android Lollipop (API Level 22): private static final
    class SingletonSet<E> extends AbstractSet<E>
    implements Serializable
```

d. Field Declarations.

57. I used the criteria in Table 2 to identify field declarations in Android that Google copied from Java. An example of a field declaration Google copied from Java is in the `ObjectStreamField.java` file in the `java.io` package on line 27 and line 47 of the Java SE 5 and API Level 9 (Android Gingerbread) source code, respectively:

```
Java SE 5: private final String name;

Android Gingerbread (API Level 9): private String name;
```

58. Another example of a field declaration Google copied from Java is in the `ObjectStreamConstants.java` file in the `java.io` package on line 22 and line 28 of the Java SE 5 source code and API Level 22 (Android Lollipop) source code, respectively:

```
Java SE 5: final static short STREAM_MAGIC =
    (short)0xaced;

Android Lollipop (API Level 22): public static final
    short STREAM_MAGIC = (short) 0xaced;
```

59. This Android line was copied from Java even though the Java declaration does not include the `public` access modifier because field declarations in an interface, such as `STREAM_MAGIC`, are implicitly `public` and the modifier was not necessary.

e. Method and Constructor Declarations.

60. I used the criteria in Table 2 to identify method and constructor declarations in Android that Google copied from Java.

61. An example of a method declaration Google copied from Java is in the

PipedReader.java file in the java.io package on line 148 and line 383 of the Java SE 5 and API Level 22 (Android Lollipop) source code, respectively:

Java SE 5: synchronized void receive(char c[], int off, int len) throws IOException

Android Lollipop (API Level 22): synchronized void receive(char[] chars, int offset, int count) throws IOException

62. This is an example of copying even though the names of the arguments to the method are different (in Java SE 5 the arguments are named `c`, `off`, and `len` whereas in Android Lollipop (API Level 22) they are named `chars`, `offset`, and `count`) because the number, order, and type of the arguments are the same.

f. Interface Declarations.

63. I used the criteria in Table 2 to identify interface declarations in Android that Google copied from Java.

64. An example of a method declaration Google copied from Java is in the Pack200.java file in the java.util.jar package on line 197 and line 86 of the Java SE 5 and API Level 22 (Android Lollipop) source code, respectively:

Java SE 5 public interface Packer

Android Lollipop (API Level 22): public static interface Packer

2. *Script Used to Identify Java Declaring Code Copied in Android*

65. I wrote a Python script called DECLARATION_PROCESSOR.py that finds declaring code within 37 specific Java API packages present in the Java SE 5 code base, as well as the Android code bases. The 37 Java API Packages are listed in Exhibit C. The script identifies the files in each of the 37 packages in the Java SE 5 source code as well as the files in these packages in the Android source code. The script then compares the filenames of each of the files within corresponding packages, for example `java.lang.reflect`, found at the directory path

\jdk1.5.0\src\java\lang\reflect\ in Java and
 \libcore\luni\src\main\java\java\lang\reflect in Android. The script then records which files in each of the 37 Java packages have the same name as the files in the corresponding Android package, found at a matching directory path as shown above. Not all of the files present in each of the 37 packages in Java SE 5 have a corresponding same name file within Android. For example, only 16 of the 22 files in the `java.lang.reflect` Java SE 5 package contain a corresponding same name file in the `java.lang.reflect` Android package.

66. The script first reads and compares the `package` and `import` statements for same name file pairs in corresponding packages in each code base, and records the `package` and `import` statements that were copied in the source code. The script then reads the Data Dictionary Report output from Understand for each of the 37 Java API Packages listed in Exhibit E for the Java and Android code bases. The script compares all entities defined by the Data Dictionary Report in the same name files in each code base and finds all instances of copied declaring code based on the criteria in Table 2 in the same name file pairs. All declaring code statements identified as copied are then written out as a “\$” delimited text file for each same name file pair as well as a “\$” delimited text file for each package. Each line in the text file includes nine pieces of information about a specific instance of declaring code that Google copied:

- a. Package – identifies which of the 37 Java packages contains the declaring code.
- b. File – identifies which .java file in which the declaring code resides.
- c. Structure – the entity within which the declaring code is contained, such as a class or interface.
- d. Type – identifies what kind of declaring code it is, including access modifier (for example public class or private method). This information is generated by Understand.
- e. Java Declaration – the declaring code that Google copied as it appears in

the Java SE 5 source code file.

- f. Java Line # – the line number in the Java SE 5 source code file where the copied declaring code resides.
- g. # Java Lines – the number of lines the declaring code occupies in the Java SE 5 source code file.
- h. Android Declaration – the declaring code that Google copied as it appears in the Android source code file.
- i. Android Line # – the line number in the Android source code file where the copied declaring code resides.

67. These pieces of information about the declaring code are separated or delimited by “\$”. The script outputs this data for each same name source code file pair found in Java and Android to a file that is named with the filename of the parsed, same name, source code files, appended with the string “_All_wSrc.txt.” The script also concatenates the text file output for each file within a package into a single text output file for each package, and this file is named with the package name appended with the string “_ALL.txt” (for example, the file for the `java.io` packages was named `java.io_ALL.txt`). The text output from the script for each package is then concatenated and imported into Microsoft Excel using “\$” as a delimiter to separate the data into columns. The Python code for this script is attached in Exhibit D.

3. *Process*

68. The purpose of my analysis was to identify and record where Google copied declaring code from Java SE 5 in Android. The steps of the process are as follows:

- 1. I was provided a zip file by Orrick containing the Oracle Java source code. I downloaded the Android API source code versions from the website <https://source.android.com/source/downloading.html> by following the instructions listed there.

I created a new project in Understand and specified Java as the programming language. I added Oracle's Java source code directories, identified in paragraph 14, named java, found at directory path L:\src\Oracle\Java 2 SEP, Ver 5\0623\OAGOOGL0100209734_HIGHLY CONFIDENTIAL - SOURCE CODE\licensebundles\jdk1.5.0\jdk-1_5_0-fcs-bin-b64-windows-i586-15_sep_2004\jdk1.5.0\src\java, and javax, found at directory path L:\src\Oracle\Java 2 SEP, Ver 5\0623\OAGOOGL0100209734_HIGHLY CONFIDENTIAL - SOURCE CODE\licensebundles\jdk1.5.0\jdk-1_5_0-fcs-bin-b64-windows-i586-15_sep_2004\jdk1.5.0\src\javax and directory path L:\src\Oracle\Java 2 SEP, Ver 5\0623\OAGOOGL0100209734_HIGHLY CONFIDENTIAL - SOURCE CODE\licensebundles\source-bundles\jdk-1_5_0-src-windows\j2se\src\share\doc\stub\javax to the Understand project. I then set up Understand to create a Data Dictionary report for Java.

I created another new project in Understand and specified Java as the programming language. I added the Android Lollipop (API Level 22) source code directories named java, found at directory path L:\src\Google\API22_LollipopMR1\libcore\luni\src\main\java\java and javax, found at directory path L:\src\Google\API22_LollipopMR1\libcore\luni\src\main\java\javax to the Understand project. I then set up Understand to create a Data Dictionary report for Android. I repeated this for Android Gingerbread (API Level 9).

2. The Data Dictionary report lists all entities for each source code file in the Understand project code base. This entity list is a superset of the declaring code. Each entry in the Data Dictionary shows the entity name and entity data type, along with the line numbers where the declaring code for each entity is found in the source code.

Under the “Configure Reports” menu selection in Understand, I set the following options for the Data Dictionary report for both of the projects described in the first step above:

- a. Java programming language
 - b. Generate a text report
 - c. Display full filenames
 - d. Display parameters
3. I then selected the “Generate Reports” option in the “Reports” menu and clicked “Generate” to create the Data Dictionary reports for both projects described in step 1.
 4. To find copied declaring code in the Java and Android API code bases, I ran the `DECLARATION_PROCESSOR.py` Python script on the Java and Android code bases using the Java and Android source code and the Understand Data Dictionary reports as input.
 5. I imported each of the text output files for an entire package into a spreadsheet, specifying the custom delimiter as ‘\$’. Where the identified declaring code was not an exact character-for-character match between Java and Android, I manually reviewed the declaring code to ensure accuracy, and then created a final formatted spreadsheet showing all copied declaring code. There were a number of reasons I would record declaring source code as copied when the statements in the source code were not character-for-character matches. For example, a class may implement the same interfaces in Java and Android, but the class declarations list them in a different order in Java and Android. Another example is a method declaration where the argument names are different between Java and Android but the number, type, and order of the arguments are the same. In my manual analysis, where I found a declaring code pair that was in fact not copied, I deleted that declaring code pair from

the list of copied declaring code in the final formatted spreadsheet. Also, on the occasion I found declaring code that was in fact copied but required a manual copy and paste of source code into the spreadsheet, I pasted in the clarifying declaring source code and noted the declaring code pair as copied in the final formatted spreadsheet. I then formatted the spreadsheet and sorted it based first on package name, then filename, and lastly the first line number of the declaring code in the Java source code.

69. I ran the process described in step 1 on the source code for Java SE 5, Android Gingerbread (API Level 9), and Android Lollipop (API Level 22). A spreadsheet summarizing the declaring code in Android Gingerbread (API Level 9) that was copied from Java SE 5 is attached as Exhibit F. A spreadsheet summarizing the declaring code in Android Lollipop (API Level 22) that was copied from Java SE 5 is attached as Exhibit S.

70. Based on my analysis, it is my opinion that Google copied the declaring code from the 37 Java API Packages in Android Gingerbread (API Level 9) and Android Lollipop (API Level 22) from Java SE 5.

4. *Presence of Private Java Declarations.*

71. Exhibits F and S each contain a notable amount of declaring code that is declared `private`. Such `private` declarations are not part of the Java API specification documentation that are publicly accessible and cannot be directly used by software developers using the 37 Java API Packages in their own software. In other words, the `private` declaring code acts in many ways like implementing code.

72. Because the `private` declaring code is not part of the public Java API, Google could not have copied this declaring code from Oracle's online Java API documentation.

73. I understand from Oracle's counsel that Oracle filed a copyright registrations for Java SE 5 (and likewise for SE 6 and SE 7) and that this is *prima facie* evidence of Oracle's ownership of the private declarations.

74. I also independently attempted to verify ownership of these private declarations. First, I looked at the copyright notice in the comments of the source code files in Java SE 5 whose private declarations Google copied in Android. These notices listed Sun or Oracle as the owner, suggesting that the declaring code belongs to Oracle.

75. Second, I attempted to eliminate the possibility that there was a common third-party source from which Java SE 5 and Android obtained the privately declared source code (in particular, code from the Apache Harmony project, where Google got much of the source code for the 37 Java API Packages). Harmony was not an Apache project until 2005, so the Java SE 5 files predate the project entirely.³ I examined the metadata and copyright and license notices for the relevant source code files in Java SE 5 and the files from Apache Harmony. The Java SE 5 files are dated September 2004, whereas the Harmony files are dated October 2007. *See* <http://archive.apache.org/dist/harmony/milestones/5.0/M3/apache-harmony-src-r580985-snapshot.zip>. Since Sun/Oracle created its files three years before Apache did, the Apache Harmony project could not be a common third party source for this code. Based on the above, I believe the logical conclusion is that Google copied this code from Sun rather than a common source third-party source.

B. Google Copied Declaring Code for the 37 Java API Packages from Java SE 5 in Other Versions of Android

76. The analysis I performed to identify the declaring code that Google copied from the 37 Java API Packages from Java SE 5 into Android, described in Section VI.A.2 above, covered two versions of Android: Android Gingerbread (API Level 9) and Android Lollipop (API Level 22).

77. It makes sense that the same declaring code should be in other versions of Android that came between Android Gingerbread (API Level 9) and Android Lollipop (API Level 22),

³ <http://harmony.apache.org/newshistory.html> (accessed Jan. 7, 2016); <http://wiki.apache.org/harmony/> (accessed Jan. 7, 2016).

and the version that came out after I started my analysis, Android Marshmallow (API Level 23).

78. Even Google acknowledges that this is how successive versions of the Android API (API Levels) relate to one another:

Updates to the framework API are designed so that the new API remains compatible with earlier versions of the API. That is, most changes in the API are additive and introduce new or replacement functionality. As parts of the API are upgraded, the older replaced parts are deprecated but are not removed, so that existing applications can still use them. In a very small number of cases, parts of the API may be modified or removed, although typically such changes are only needed to ensure API robustness and application or system security. All other API parts from earlier revisions are carried forward without modification.

See <http://developer.android.com/guide/topics/manifest/uses-sdk-element.html>.

79. To test the proposition that the declaring code from the 37 Java API Packages was also in these other versions of Android, I took two approaches: (1) checking Android source code for the declaring code I had already determined that Google copied in API Level 22 (Android Lollipop) and API Level 9 (Gingerbread); and (2) examining Google's public statements on changes to the 37 Java API Packages as they appear in Android. Both approaches confirmed that Google copied the declaring code for the 37 Java API Packages into these versions of Android.

1. *Source Code Analysis of Other Versions of Android*

80. My analysis of the source code from Android Lollipop (API Level 22), described in Section VI.A.3 above, identified instances where the declaring code from Java SE 5 was copied in Android Lollipop (API Level 22). My analysis of the source code from Android Gingerbread (API Level 9), described in Section VI.A.3 above, identified instances where the declaring code from Java SE 5 was copied in Android Gingerbread (API Level 9).

81. I wrote a python script named `AndroidDeclarationDiff.py`, and included as Exhibit E, that reads the copied declaring code I found in Android Gingerbread (API Level 9) and Android Lollipop (API Level 22) and then checks to see if that same declaring code is present in other versions of Android, indicating it was copied in these other versions of Android. I ran this

script to search for the copied declaring code from Android Gingerbread (API Level 9) and Android Lollipop (API Level 22) within source code files that have the same name and are in the same directory location in other versions of Android.

82. The other versions of Android I analyzed include Android API Level 10, Android API Level 14, Android API Level 15, Android API Level 16, Android API Level 17, Android API Level 18, Android API Level 19, Android API Level 20, Android API Level 21, and Android API Level 23.

83. The script runs on one version of Android source code at a time. For every line of declaring code I previously found copied in Android Gingerbread (API Level 9) and Android Lollipop (API Level 22), the script looks for that exact⁴ line of declaring code in the other version of Android I am analyzing. If the exact line of declaring code is present, I record that line and conclude that the line of declaring code was also copied in the other Android version. However if the exact line of declaring code is not found, I keep track of it for further review.

84. Next, I manually analyzed those lines of declaring code from Android Gingerbread (API Level 9) and Android Lollipop (API Level 22) for which I did not find an exact match in the other version of Android I am analyzing. To do this, I looked to see if there was source code in the other version of Android being analyzed that I could conclude was copied based on the criteria in Table 2. Where I found such instances of copying, I recorded them in my output file.

85. At the end of this analysis, I had a list of the lines of declaring code that were copied from Java SE 5 in that version of Android.

86. Based on this analysis, I identified 11,740 declaring code statements in Android API Level 10 that were copied from Java SE 5. *See* Exhibit G.

87. Based on this analysis, I identified 11,011 declaring code statements in Android API Level 14 that were copied from Java SE 5. *See* Exhibit K.

88. Based on this analysis, I identified 11,011 declaring code statements in Android API

⁴ As part of this analysis, I checked for exact matches ignoring whitespace and newline characters.

Level 15 that were copied from Java SE 5. *See* Exhibit L.

89. Based on this analysis, I identified 10,806 declaring code statements in Android API Level 16 that were copied from Java SE 5. *See* Exhibit M.

90. Based on this analysis, I identified 11,012 declaring code statements in Android API Level 17 that were copied from Java SE 5. *See* Exhibit N.

91. Based on this analysis, I identified 11,038 declaring code statements in Android API Level 18 that were copied from Java SE 5. *See* Exhibit O.

92. Based on this analysis, I identified 11,045 declaring code statements in Android API Level 19 that were copied from Java SE 5. *See* Exhibit P.

93. Based on this analysis, I identified 11,030 declaring code statements in Android API Level 20 that were copied from Java SE 5. *See* Exhibit Q.

94. Based on this analysis, I identified 11,049 declaring code statements in Android API Level 21 that were copied from Java SE 5. *See* Exhibit R.

95. Based on this analysis, I identified 11,009 declaring code statements in Android API Level 23 that were copied from Java SE 5. *See* Exhibit T.

2. *Google's Public Documents Confirm Copying Exists in Other Versions of Android*

96. Google generates “Android API Differences Reports” (“Diff Reports”) and makes these reports publicly available on its Android Developers website, <http://developer.android.com>. These reports identify changes to the APIs available in Android between successive versions. As Google itself describes the reports:

This report details the changes in the core Android framework API between two API Level specifications. It shows additions, modifications, and removals for packages, classes, methods, and fields. The report also includes general statistics that characterize the extent and type of the differences.

This report is based a comparison of the Android API specifications whose API Level identifiers are given in the upper-right corner of this page. It compares a newer “to” API to an older

“from” API, noting all changes relative to the older API. So, for example, API elements marked as removed are no longer present in the “to” API specification.⁵

97. Having analyzed the source code from API Level 9 (Android Gingerbread) and API Level 22 (Android Lollipop) already, I focused this analysis on other versions of Android, namely API Levels 10–21, and 23. Table 3 identifies the Diff Reports I considered in my analysis. The API Level in the “To” column is version of Android that is the subject of the report and being compared to the version in the “From” column.

Android API Level Comparison		URL for Diff Report
To	From	
10	9	https://developer.android.com/sdk/api_diff/10/changes.html
11	10	https://developer.android.com/sdk/api_diff/11/changes.html
12	11	https://developer.android.com/sdk/api_diff/12/changes.html
13	12	https://developer.android.com/sdk/api_diff/13/changes.html
14	13	https://developer.android.com/sdk/api_diff/14/changes.html
15	14	https://developer.android.com/sdk/api_diff/15/changes.html
16	15	https://developer.android.com/sdk/api_diff/16/changes.html
17	16	https://developer.android.com/sdk/api_diff/17/changes.html
18	17	https://developer.android.com/sdk/api_diff/18/changes.html
19	18	https://developer.android.com/sdk/api_diff/19/changes.html
20	19	https://developer.android.com/sdk/api_diff/20/changes.html
21	20	https://developer.android.com/sdk/api_diff/21/changes.html
22	21	https://developer.android.com/sdk/api_diff/22/changes.html
23	22	https://developer.android.com/sdk/api_diff/23/changes.html

Table 3. Diff Reports considered in my analysis

98. For API Levels 19 through 23, the URL in Table 3 is a webpage comprising a Diff Report. For example, the Diff Report comparing API Level 19 to API Level 18 (available at https://developer.android.com/sdk/api_diff/19/changes.html) looks like:

⁵ See, e.g., https://developer.android.com/sdk/api_diff/19/changes.html.

The screenshot displays the 'Android API Differences Report' for API levels 18 to 19. The left sidebar provides navigation options to filter differences by package, class, constructor, method, or field. The main content area explains that the report details changes in the core Android framework API, including additions, modifications, and removals. It also provides instructions on how to navigate the report using the sidebar controls. A table titled 'Added Packages' lists several new packages added in API level 19, including `android.graphics.pdf`, `android.nfc.cardemulation`, `android.print`, `android.print.pdf`, `android.printservice`, and `android.transition`. The top right corner shows the API diff specification details: To Level: 19, From Level: 18, Generated: 2013.10.29 16:44.

99. For API Levels 9–18, the above URL is a link to a compressed archive file (.tar.gz) that contains the Diff Report. When extracted, the compressed archive creates the files for a webpage comprising the Diff Report that looks, behaves, and functions like those webpage Diff Reports mentioned in paragraph 98, above.

100. The Diff Report identifies the changes in the Android APIs in a number of ways, including by package. This means a user of a Diff Report can identify which API packages have been changed since the last API Level. This view is accessible by clicking the link for “By Package” from the upper-left frame of the Diff Report.

The screenshot shows the 'Android API Differences Report' page. The 'Select a Diffs Index:' panel on the left has 'By Package' selected and highlighted with a red box. The 'Filter the Index:' panel shows 'All Differences' selected. Below these panels is a list of package names starting with 'A' through 'Z'. The main content area contains an introduction to the report, navigation instructions, and a table titled 'Added Packages'.

Added Packages	
android.graphics.pdf	
android.nfc.cardemulation	
android.print	
android.print.pdf	
android.printservice	
android.transition	

101. Using this By Package view, I was able to determine, for a given Diff Report, which of the 37 Java API Packages at issue had changed. I did this by looking in the lower-left frame of the Diff Report and looking for package names that were among the 37 Java API Packages listed in Exhibit C. By repeating this process for each Diff Report in Table 3, I identified the versions of Android, by API Level, that made changes to the 37 Java API Packages and which packages were affected.

102. Where a given Diff Report does not list any of the 37 Java API Packages as having changed, I concluded that Google is accurately reporting that there were no changes and that the version of Android contains the same copied declaring code as the previous version of Android. For example, the Diff Report for API Level 10 does not identify any of the 37 Java API Packages as having changed. Therefore, I conclude that the declaring code for the 37 Java API Packages in API Level 10 was copied in the same way as in API Level 9.

103. Table 4 summarizes which API Levels the Diff Reports identify as having changed for the 37 Java API Packages, and which of these packages were affected.

Android API Level	37 Java API Packages Changed
API Level 10	<i>None</i> ; declaring code same as API Level 9.
API Level 11	java.lang java.util
API Level 12	<i>None</i> ; declaring code same as API Level 11.
API Level 13	<i>None</i> ; declaring code same as API Level 11.
API Level 14	java.io java.lang java.lang.ref java.lang.reflect java.net java.security java.util.logging javax.security.auth
API Level 15	<i>None</i> ; declaring code same as API Level 14.
API Level 16	<i>None</i> ; declaring code same as API Level 14.
API Level 17	<i>None</i> ; declaring code same as API Level 14.
API Level 18	<i>None</i> ; declaring code same as API Level 14.
API Level 19	java.io java.lang java.lang.reflect java.net java.nio java.nio.channels java.nio.charset java.sql java.util java.util.logging java.util.zip javax.crypto javax.crypto.spec
API Level 20	<i>None</i> ; declaring code same as API Level 19.
API Level 21	java.util java.util.zip
API Level 22	<i>None</i> ; declaring code same as API Level 21.
API Level 23	<i>None</i> ; declaring code same as API Level 21.

Table 4. API Levels the Diff Reports identify as having changed

104. According to the Diff Reports for Android versions corresponding to API Levels 11, 14, 19, and 21, these versions of Android contain changes to the 37 Java API Packages. I inspected these Diff Reports to determine if the changes affect my opinions about the declaring

code that Google copied from Java SE 5 into Android. They did not. The changes to the 37 Java API Packages for these versions of Android are attached as Exhibits Z (API Level 11), AA (API Level 14), BB (API Level 19), and CC (API Level 21).

105. The changes to the 37 Java API Packages in Exhibits Y through CC included additions (*e.g.* new classes or methods) and changes (*e.g.* changing a method from being abstract to non-abstract), and removals (*e.g.* removing a method from a class).

106. Where there were *additions* to the packages, these changes are not relevant to whether the declaring code in Android was copied from Java SE 5, though it may be relevant to copying from later versions of Java. *See* Section VI.D, below.

107. Where there were *changes* to the packages, I looked to see if the specific change suggested that Google did not copy the declaring code. None of these changes alters my opinion that Google copied declaring code the 37 Java API Packages from Java SE 5 in Android.

3. Google Copied Declaring Code from Java SE 5 in Android Honeycomb.

108. As described in Paragraph 30, above, I was unable to analyze the Android Honeycomb (API Levels 11 through 13) source code, however I am able to identify the declaring code Google copied using the results from my analysis of Android Gingerbread (API Level 10) and Google's Diff Reports.

109. Based on my earlier analysis, I identified 11,723 lines of declaring code in Android Gingerbread (API Level 10) that were copied from Java SE 5. *See* Exhibit G.

110. The Diff Report for Android Honeycomb (API Level 11) identified changes to or additions of 29 lines of declaring code in `java.lang` and `java.util`. I confirmed that none of these changes were present in Java SE 5. On this basis, it is my opinion that there are 11,723 lines of declaring code in Android Honeycomb (API Level 11) that were copied from Java SE 5. *See* Exhibit H.

111. In the Diff Reports for Android Honeycomb (API Level 12) and Android Honeycomb (API Level 13), Google claims that it did not make any changes to the 37 Java API Packages. On

this basis, it is my opinion that the number of lines of declaring code copied from Java SE 5 for these versions of Android is the same as Android Honeycomb (API Level 11): 11,723 lines. *See* Exhibits I, J.

C. Google Copied the SSO of the 37 Java API Packages From Java SE 5 in Android

112. A jury already found that Google copied the SSO of the 37 Java API Packages from Java SE 5 in versions of Android through Android Froyo (API Level 8). In my opinion, Google has continued to copy the SSO in Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23).

113. My previous analyses in Section VI.A.3 identifying lines of declaring code that Google copied from Java SE 5 in Android Gingerbread (API Level 9) onward taken in full context also indicates that Google copied the SSO of Java SE 5 (*e.g.* which package a class belongs to, which methods and fields are defined in that class, etc.).

114. By copying declaring code from the 37 Java API Packages from Java SE 5, Google copied not only literal lines of code, but also the intricate relationships between and among the classes, methods, interfaces, and fields in those packages. Trial Exhibit 1028 identifies the Java API Packages, classes, interfaces, and the relationships between and among them, illustrating the SSO of Java SE 5. Google copied this SSO in Android Gingerbread through Marshmallow.

115. I also observed Google copying the SSO of the 37 Java API Packages when I did a visual comparison of the API Documentation for Java SE 5 and Android Lollipop (API Level 22). Both Java and Android document their respective API Packages in online documentation. The Java SE 5 documentation is available at <http://docs.oracle.com/javase/1.5.0/docs/api>. The Android API documentation is available at <http://developer.android.com/reference/packages.html>.

116. The documentation reflects both the SSO of Java SE 5, as well as the declaring code contained in the source code files comprising the packages. For example, the documentation identifies the classes, interfaces, exceptions, enums, errors, and annotation types that are

associated with a package.

117. I visually inspected the Java and Android documentation for each package in the 37 Java API Packages. I observed that the classes, interfaces, exceptions, enums, errors, and annotation types are organized in the same hierarchical manner, within packages of the same name, in Android as they occur in Java. Exhibit U shows a side-by-side comparison showing this identical hierarchical organizational structure for each of the 37 Java API Packages.

118. My visual comparison of the SSO of Java SE 5, which is expressed through this online documentation, shows the same hierarchical organization for the classes.

119. From the analysis above, I conclude that Google's copying of the declaring code for the 37 Java API Packages into Android also led to copying of the SSO for Java SE 5.

D. Google Has Continued to Copy New Declaring Code and SSO of the 37 Java API Packages from Java SE 6 and Java SE 7

120. In addition to copying declaring code for the 37 Java API Packages from Java SE 5, Google has also copied declaring code from later versions of Java: Java Platform, Standard Edition 6 ("Java SE 6") and Java Platform, Standard Edition 7 ("Java SE 7").

121. I understand Java SE 6 was released in December 2006 and Java SE 7 was released in July 2011. Thus at least some of the copying of the declaring code from Java SE 6 and Java SE 7 occurred after Oracle filed this lawsuit in August 2010.

122. I arrived at this conclusion by further examining the results of my analysis of the Android Diff Reports as described in Paragraphs 96–107. For each change to the 37 Java API Packages in Android, I looked to see when that change appeared in a version of Java. I did this by looking at the online API documentation for Java.⁶

123. For example, in the Diff Report for Android Kit Kat (API Level 19), it identifies class

⁶ In particular, I looked at documentation for Java SE 1.3.1 (<https://docs.oracle.com/javase/1.3/docs/api/>, download required), Java SE 1.4.2 (<https://docs.oracle.com/javase/1.4.2/docs/api/>, download required), Java SE 5 (<https://docs.oracle.com/javase/1.5.0/docs/api/>), Java SE 6 (<http://docs.oracle.com/javase/6/docs/api/>), Java SE 7 (<http://docs.oracle.com/javase/7/docs/api/>), and Java SE 8 (<http://docs.oracle.com/javase/8/docs/api/>).

GCMParameterSpec being added to package `javax.crypto.spec`. The online documentation for Java SE 7 identifies this class as being added in Java SE 7 and it is not present in the Java SE 6 online documentation. Therefore, I can conclude that this class (and the related constructors and methods) were introduced in Java SE 7.

124. Based on my analysis, I identified 816 instances of Google copying declaring code introduced in Java SE 6 into Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23). A summary of my findings for Java SE 6 is attached as Exhibit V. I also identified 223 instances of Google copying declaring code introduced in Java SE 7 into Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23). A summary of my findings for Java SE 7 is attached as Exhibit W.

125. By copying the declaring code from Java SE 6 and Java SE 7 in these versions of Android, Google also copied the intricate relationships of the classes, methods, interfaces, and fields in those packages, thus copying the SSO of Java SE 6 and Java SE 7.

E. Google Copied the Declaring Code and SSO of the 37 Java API Packages in Google's "ARC" Product

1. *The App Runtime for Chrome ("ARC") Technology Allows Android Apps to Run on Chrome OS Devices*

126. The App Runtime for Chrome (ARC) is a runtime environment created by Google that allows the Android runtime to be used on Chrome OS devices.⁷ One of the primary uses for this feature is the ability to run Android apps on the Chrome OS platform.

127. In order to use ARC to run Android apps, an app called ARC Welder is used. ARC Welder allows APK files⁸ to be repacked such that they can be used with the ARC technology.⁹

⁷ See https://developer.chrome.com/apps/getstarted_arc for a brief overview of ARC.

⁸ APK files contain all parts of an Android app in a single file, and can be run directly on the Android operating system.

Once ARC Welder has repackaged the APK file, the Android app can be run using a Chrome OS device with ARC installed. Note that ARC refers to the actual runtime, while ARC Welder is simply the repackaging tool.

128. It is important to note that ARC Welder does not actually convert the resources from the Android app from one format into another or transform the app in any way, it simply reorganizes the app's existing resources such that they can be accessed and executed by Chrome OS.

129. Along with ARC, Google also released a set of four apps to the Chrome Web Store that were converted from Android apps in the Google Play Store. These four apps are Duolingo, Evernote, Sight Words, and Vine.^{10 11} These apps, along with other Android apps, are repackaged and available for download and use on Chrome, and are referred to on the Chrome Web Store as "ARC apps."

130. The ARC technology is separate from the ARC Welder tool. When the ARC Welder tool is installed onto a Chrome OS device, the installation process checks to see whether ARC is installed, and automatically installs ARC if it is not. The same ARC installation check takes place when downloading any ARC app from the Chrome Web Store.

2. *The ARC Runtime Introduces Android Runtime Components into Chrome OS Devices*

131. Since a Chrome OS device with ARC installed is able to run Android apps, this implies that there must be certain runtime software components present in a Chrome OS device

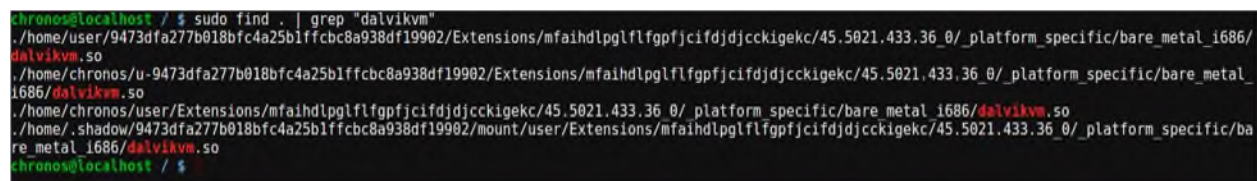
⁹ See <https://chrome.google.com/webstore/detail/arc-welder/emfinbmielocnlhgmfkkmkngdoccbadn> for the download page for the ARC Welder app on the Google Play Store.

¹⁰ See <http://chrome.blogspot.com/2014/09/first-set-of-android-apps-coming-to.html> for the Chrome Blogspot posting announcing the release of the first four Android apps packaged for use on Chrome and released to the Chrome Web Store.

¹¹ See <https://chrome.google.com/webstore/detail/duolingo/ebnhfamfopiobpaehmebmfcgkaogihe?hl=en-US> (Duolingo), <https://chrome.google.com/webstore/detail/evernote/dhfolfjkpaeajbiicgheljefkfbfbke?hl=en-US> (Evernote), <https://chrome.google.com/webstore/detail/sight-words/ikmpccnfemdkmmoejgmdajnkbidifpgh?hl=en> (Sight Words), and <https://chrome.google.com/webstore/detail/vine/plfjlfhofhjipmmifkbcmalnmcebkklkh?hl=en-US> (Vine) for the pages on the Chrome Web Store from which these ARC apps can be downloaded.

with ARC that are the same as those used by the Android operating system.

132. Searching through the file system of an ARC-enabled Chromebook running the latest version of Chrome OS¹² revealed that binary files with names referencing the Dalvik virtual machine and core Java classes are present in the Chromebook file system. These files are `dalvikvm.so`, `libjavacore.so`, and `libjavacrypto.so`. See Figure 1 for the results of a search through the file system, which returns results showing binary files that appear to be associated with the Dalvik virtual machine.



```

chronos@localhost / $ sudo find . | grep "dalvikvm"
./home/user/9473dfa277b018bfc4a25b1ffc8a938df19902/Extensions/mfaihdlpglflfgpfjcidjdjckigekc/45.5021.433.36_0/_platform_specific/bare_metal_i686/dalvikvm.so
./home/chronos/u-9473dfa277b018bfc4a25b1ffc8a938df19902/Extensions/mfaihdlpglflfgpfjcidjdjckigekc/45.5021.433.36_0/_platform_specific/bare_metal_i686/dalvikvm.so
./home/chronos/user/Extensions/mfaihdlpglflfgpfjcidjdjckigekc/45.5021.433.36_0/_platform_specific/bare_metal_i686/dalvikvm.so
./home/.shadow/9473dfa277b018bfc4a25b1ffc8a938df19902/mount/user/Extensions/mfaihdlpglflfgpfjcidjdjckigekc/45.5021.433.36_0/_platform_specific/bare_metal_i686/dalvikvm.so
chronos@localhost / $

```

Figure 1: Shared object files from the Dalvik Virtual Machine found in the file system of a Chromebook device indicate the presence of Android runtime components.

133. The contents of these `.so` files are low-level object code that is not human-readable by direct inspection. In order to make sense of the contents of these binaries, the `readelf` command was used in order to interpret contents of these binary files.¹³

134. A selected portion of the output of the `readelf` command for the `libjavacore.so` file is shown in Figure 2. The full `readelf` output for this binary file is presented in Exhibit X. Each row of the output provides detailed information about a single symbol in the ELF file. The rightmost part of the entry gives the name of the symbol itself. From the information in the figure, it is clearly evident that the names of these symbols are virtually identical to the names of some of the infringed packages (`java.io`, `java.lang`, `java.nio`, `java.text`, and

¹² This test used an Acer Chromebook 11, running in developer mode and using the 11 December 2015 build of Chrome OS. See https://store.google.com/product/acer_chromebook_11 for the product description page of this model on the Google Store.

¹³ The `readelf` command is a GNU binary utility used to display information about ELF (Executable and Linkable Format) object files, a common file format for object code and shared libraries. See <https://sourceware.org/binutils/docs/binutils/readelf.html> for a detailed description of the `readelf` command. Since ELF-type files (such as `.so` files) are not human-readable, the `readelf` command provides a means by which the contents of these types of files can be displayed in a human-readable manner. In this case, the `readelf -all` command was used, which lists the most complete information about the contents of the ELF files.

java.util).

135. This shows that the file contains compiled binary versions of classes derived from the 37 Java API Packages at issue. Because such binary files are present in the file system of the Chromebook following installation of ARC technology, the ARC runtime environment makes use of the infringing material in order to allow Android applications to be run on Chrome OS devices.

51:00:00 00017f24	65 FUNC	GLOBAL	DEFAULT	8 _Z21register_java_io_File
52:00:00 00018be0	65 FUNC	GLOBAL	DEFAULT	8 _Z31register_java_io_File
53:00:00 00018ed4	65 FUNC	GLOBAL	DEFAULT	8 _Z34register_java_io_Obj
54:00:00 19560	65 FUNC	GLOBAL	DEFAULT	8 _Z28register_java_lang_Ch
55:00:00 000195c0	65 FUNC	GLOBAL	DEFAULT	8 _Z25register_java_lang_Do
56:00:00 19614	65 FUNC	GLOBAL	DEFAULT	8 _Z24register_java_lang_Fl
57:00:00 00019a54	65 FUNC	GLOBAL	DEFAULT	8 _Z23register_java_lang_Ma
58:00:00 00019e1c	65 FUNC	GLOBAL	DEFAULT	8 _Z33register_java_lang_Pr
59:00:00 0001aac8	65 FUNC	GLOBAL	DEFAULT	8 _Z31register_java_lang_Re
60:00:00 0001aee0	65 FUNC	GLOBAL	DEFAULT	8 _Z29register_java_lang_St
61:00:00 0001c994	65 FUNC	GLOBAL	DEFAULT	8 _Z31register_java_lang_St
62:00:00 0001cd94	65 FUNC	GLOBAL	DEFAULT	8 _Z25register_java_lang_Sy
63:00:00 00021c68	65 FUNC	GLOBAL	DEFAULT	8 _Z27register_java_math_Na
64:00:00 00021cb4	65 FUNC	GLOBAL	DEFAULT	8 _Z27register_java_nio_Byt
65:00:00 00022fb4	65 FUNC	GLOBAL	DEFAULT	8 _Z34register_java_nio_cha
66:00:00 23744	65 FUNC	GLOBAL	DEFAULT	8 _Z23register_java_text_Bi
67:00:00 00023ccc	65 FUNC	GLOBAL	DEFAULT	8 _Z36register_java_util_ja
68:00:00 24968	65 FUNC	GLOBAL	DEFAULT	8 _Z32register_java_util_re
69:00:00 00024d90	65 FUNC	GLOBAL	DEFAULT	8 _Z32register_java_util_re
70:00:00 00024edc	65 FUNC	GLOBAL	DEFAULT	8 _Z30register_java_util_zi
71:00:00 0002501c	65 FUNC	GLOBAL	DEFAULT	8 _Z28register_java_util_zi
72:00:00 25634	65 FUNC	GLOBAL	DEFAULT	8 _Z31register_java_util_zi
73:00:00 00025d74	65 FUNC	GLOBAL	DEFAULT	8 _Z31register_java_util_zi

Figure 2: Certain symbols from the `readelf` output of binary file `libjavacore.so` contain compiled resources from the 37 infringed packages.

3. *The ARC Runtime Build Process Contains the 37 Java API Packages.*

136. The publicly available code repository for ARC Runtime contains a folder called `third_party/android/libcore` folder¹⁴ that links directly to the called

¹⁴ The libcore folder for ARC Runtime can be accessed at https://chromium.googlesource.com/arc/arc/+/refs/tags/arc-runtime-48.5021.561.0/third_party/android/

platform/libcore folder in in Android Open Source Project Repository¹⁵, as shown in Figure 3, which contains the Android source code for the 37 Java API Issues.

[chromium / arc / arc / refs/tags/arc-runtime-48.5021.561.0 / . / third_party / android / libcore](https://chromium.googlesource.com/arc/arc/refs/tags/arc-runtime-48.5021.561.0/.?third_party/android/libcore)
 Submodule link to 6632d8c9d8d1a3ac338d541676148677641baf3 of <https://android.googlesource.com/platform/libcore>

Figure 3. Libcore folder in the publicly available code repository for ARC runtime

137. Furthermore, the build scripts for the ARC Runtime references, also found in the publicly available code repository, references the code that contains the 37 Java API Packages in the third_party/android/libcore folder.

4. *Android Apps Made Available for Use on Chrome through ARC Rely on and Use the 37 Java API Packages*

138. As was mentioned before, Evernote was one of the four original ARC apps released to the Chrome Web Store as Android apps repackaged and ready for use on Chrome OS. Evernote is one of the most frequently downloaded Android apps available on the Google Play Store, having been downloaded to hundreds of millions of Android devices to date.¹⁶

139. The Evernote app architecture implements material from the 37 Java API packages. The advent of ARC and ARC apps makes the Evernote app, as well as its leveraging of infringed content, available for use on Chrome OS devices. To this end, an analysis was conducted to determine which of the 37 Java API packages are used in Evernote.

140. To perform this analysis, the Evernote Android APK file was downloaded from the Google Play Store using the Firefox APK Downloader plugin.¹⁷ Following this, a series of reverse-engineering tools were used to convert the apk file back into source code such that its

¹⁵ The libcore folder in AOSP can be accessed at <https://android.googlesource.com/platform/libcore>

¹⁶ This was determined by retrieving a dataset showing user rating and download metrics for all apps available on the Google Play Store. Based on this data, the Evernote app reports having somewhere between 100 million and 500 million app downloads as of 21 December 2015.

¹⁷ See <https://play.google.com/store/apps/details?id=com.evernote&hl=en> for the Google Play Store page for the Evernote app.

underlying dependencies could be extracted.¹⁸ A tool called Understand was then used to extract the dependency structure of the app from the reverse-engineered source code.¹⁹

141. The results of the analysis show the Evernote app using 13 of the 37 Java API packages in its source code: `java.net`, `java.nio`, `java.lang.reflect`, `javax.net`, `java.lang.ref`, `java.io`, `java.security`, `java.text`, `java.util.zip`, `java.util.regex`, `java.lang`, `java.util`, and `javax.crypto`.

142. This shows that ARC makes available to Chrome OS devices an app that has been downloaded hundreds of millions of times, and leverages the functionality of 13 of the 37 Java API packages through use of the Android runtime.

143. In a broader sense, ARC expands the extent to which the Android platform benefits from its appropriation of the 37 infringed packages. It does so both by actually implementing the infringed content in the ARC technology itself, as well as by providing a medium of access by which Chrome OS devices can use app content that leverages this infringed material as well.

VII. CONCLUSION

144. I have analyzed the source code for both Java SE 5 and Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23), looking for instances where declaring code for the 37 Java API Packages from Java SE 5 appears in Android. Based on this analysis, I conclude the following:

- a. Android Gingerbread (API Level 9) contains 11,723 declaring code statements from Java SE 5;
- b. Android Gingerbread (API Level 10) contains 11,723 declaring code statements from Java SE 5;

¹⁸ This process begins with simply unpacking the APK file to extract its contents, and using a tool called dex2jar (<https://github.com/pxb1988/dex2jar>) to convert the .dex bytecode contents of the APK file back into .class bytecode format. Note that this does not transform the contents in any functional manner, but simply converts from one format to another. After this, the output of the dex2jar tool is processed by a tool called CFR (<http://www.benf.org/other/cfr/>) which de-compiles the bytecode into its original Java source code.

¹⁹ See <https://scitools.com/> for a description of the various functions of the Understand static code analysis tool.

- c. Android Honeycomb (API Level 11) contains 11,723 declaring code statements from Java SE 5;
- d. Android Honeycomb (API Level 12) contains 11,723 declaring code statements from Java SE 5;
- e. Android Honeycomb (API Level 13) contains 11,723 declaring code statements from Java SE 5;
- f. Android Ice Cream Sandwich (API Level 14) contains 11,692 declaring code statements from Java SE 5;
- g. Android Jelly Bean (API Level 15) contains 11,692 declaring code statements from Java SE 5;
- h. Android Jelly Bean (API Level 16) contains 11,693 declaring code statements from Java SE 5;
- i. Android Jelly Bean (API Level 17) contains 11,694 declaring code statements from Java SE 5;
- j. Android Jelly Bean (API Level 18) contains 11,720 declaring code statements from Java SE 5;
- k. Android Kit Kat (API Level 19) contains 11,730 declaring code statements from Java SE 5;
- l. Android Kit Kat (API Level 20) contains 11,731 declaring code statements from Java SE 5;
- m. Android API Lollipop (Level 21) contains 11,734 declaring code statements from Java SE 5;
- n. Android Lollipop (API Level 22) contains 11,735 declaring code statements from Java SE 5;
- o. Android Marshmallow (API Level 23) contains 11,717 declaring code statements from

Java SE 5;

145. I further conclude that Google copied the SSO for the 37 Java API Packages in Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23).

146. I also conclude that Google copied 816 lines of declaring code from Java SE 6 and 223 lines of declaring code from Java SE 7 in Android Gingerbread (API Level 9) through Android Marshmallow (API Level 23).

147. Finally, I conclude that by enabling Android apps to be run on Chrome OS devices such as Chromebooks, Google has copied the declaring code and SSO of the 37 Java API Packages.

148. It is my understanding that discovery in this case is ongoing. Accordingly, I reserve the right to supplement or amend my opinions in light of any additional evidence, testimony, or information that may be provided to me after the date of this report. I also reserve the right to supplement or amend my opinions in response to any expert reports served by any party in the lawsuit.

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

Executed on: January 8, 2016

A handwritten signature in black ink, appearing to read "Robert Zeidman", written over a horizontal line.

Robert Zeidman

PROOF OF SERVICE BY KITEWORKS

I, José E. Valdés, am over the age of eighteen years old and not a party to the within-entitled action. My place of employment and business address is Orrick, Herrington & Sutcliffe LLP, 1000 Marsh Road, Menlo Park, California 94025.

On January 8, 2016, I served the following documents:

EXPERT REPORT OF ROBERT ZEIDMAN.

on the interested parties in this action by electronic service [Fed. Rule Civ. Proc. 5(b)] by electronically mailing a true and correct copy, pursuant to the parties agreement, to the following email addresses:

DALVIK-KVN@kvn.com
JCooper@fbm.com
gglas@fbm.com

I declare under penalty of perjury under the laws of the State of California and the United States that the foregoing is true and correct.

Executed on January 8, 2016, at San Francisco, California.

/s/ José E. Valdés
José E. Valdés